SymPy の多項式 GCD 計算の改善

Optimizing Polynomial GCD Algorithm in SymPy

神戸大学 大学院 人間発達環境学研究科　　松林 龍世　　長坂 耕作 [*1]
Ryusei Matsubayashi　Kosaku Nagasaka
Graduate School of Human Development and Environment, Kobe University

Abstract

This study introduces a novel enhancement to the GCDHEU algorithm by leveraging the degree difference between two polynomials. By strategically modifying the variable substitution order, our method delivers significant improvements in computational time. To support this improvement, we also detail the heuristic components and operational principles underlying GCDHEU.

1　はじめに

本報告では，SymPy 上における GCDHEU アルゴリズムの改善を取り扱っている。SymPy では多項式 GCD に関して，Heuristic GCD[CGG84], Modular GCD[MW00], PRS GCD[Bro78] などのアルゴリズムが実装[1)]されており，そのなかでも主に Heuristic GCD（以後，GCDHEU と記載）が用いられている。GCDHEU アルゴリズムは，高速算法として知られている Zippel[Zip79] の方法などに比べてシンプルで実装しやすく簡単である。多くの人が触れる機会のある SymPy でも用いられていることから，GCDHEU アルゴリズムの改善にも一定の意義があると考えたのが本報告の背景である。あくまでも，その特徴である「簡単でシンプルな状態」を維持した上での改善が目的であり，最速の多項式 GCD アルゴリズムを目指すものではないという点についてはご留意願いたい。

2　GCDHEU

整数環 $\mathbb{Z}$ 上の k 変数多項式環を $\mathbb{Z}[x_1, \ldots, x_k]$ とし，$\mathbb{Z}[\vec{x}]$ と略記する。GCDHEU は，$\mathbb{Z}[\vec{x}]$ の二つの多項式の最大公約因子（以後，GCD）を求めるアルゴリズムの一つである。確率的アルゴリズムのため，正しい結果を得るまで逐次的に正誤判定が内部で必要とされる。変数の個数が増えると劇的に遅くなるものの，4 変数程度までは効果的であるという先行研究 [LF95] がある。SymPy, Maple, SageMath(FLINT) などの実装には多少の差異は見られるが，基本的にはアルゴリズム 1 と大きく違いはない。なお，多項式 $p(\vec{x}) \in \mathbb{Z}[\vec{x}]$ に対して，$\mathrm{pp}(p(\vec{x}))$ で原始的部分を，$\mathrm{cont}(p(\vec{x}))$ で係因数を表すものとし，$\zeta \in \mathbb{Z}$ に対して，$\mathrm{rem}(p(\vec{x}), \zeta)$ で ζ による剰余を表すものとする（剰余は，$p(\vec{x})$ の各整数係数 $c \in \mathbb{Z}$ に対して，$-\frac{1}{2}\zeta < \mathrm{rem}(c, \zeta) \leq \frac{1}{2}\zeta$ を満たすようにする）。

[*1] E-mail: nagasaka@main.h.kobe-u.ac.jp

[1)] GCDHEU 以外は，SymPy のソースコードに記載されている参考文献の情報に基づく。

アルゴリズム 1 Heuristic GCD (GCDHEU)

入力: $f(\vec{x}), g(\vec{x}) \in \mathbb{Z}[\vec{x}] = \mathbb{Z}[x_1, \ldots, x_k]$ （ただし，$f, g \in \mathbb{Z}$ のときは $k = 0$ とみなす）
出力: $\gcd(f, g) \in \mathbb{Z}[\vec{x}]$

1: **if** $k = 0$ **then**
2: **return** $\gcd(f, g) \in \mathbb{Z}$; （GCD 計算は，整数環での Euclid の互除法などによる）
3: **end if**
4: $\alpha \leftarrow \gcd(\mathrm{cont}(f(\vec{x})), \mathrm{cont}(g(\vec{x})))$; $f(\vec{x}) \leftarrow \mathrm{pp}(f(\vec{x}))$; $g(\vec{x}) \leftarrow \mathrm{pp}(g(\vec{x}))$;
5: $\zeta \leftarrow 2 \cdot \min(\|f(\vec{x})\|_\infty, \|g(\vec{x})\|_\infty) + 2$;
6: **while** true **do**
7: $\gamma \leftarrow \mathrm{GCDHEU}\left(f(\vec{x})|_{x_1=\zeta}, g(\vec{x})|_{x_1=\zeta}\right)$; $h(\vec{x}) \leftarrow 0$;
8: **for** $i = 0, 1, \ldots$ **while** $\gamma \neq 0$ **do**
9: $h_i \leftarrow \mathrm{rem}(\gamma, \zeta)$; $h(\vec{x}) \leftarrow h(\vec{x}) + h_i \cdot x_1^i$; $\gamma \leftarrow (\gamma - h_i)/\zeta$;
10: **end for** （剰余は，γ の各整数係数 c が，$-\frac{1}{2}\zeta < \mathrm{rem}(c, \zeta) \leq \frac{1}{2}\zeta$ を満たすようにする）
11: **if** $\mathrm{pp}(h(\vec{x})) \mid f(\vec{x})$ **and** $\mathrm{pp}(h(\vec{x})) \mid g(\vec{x})$ **then**
12: **return** $\alpha \cdot \mathrm{pp}(h(\vec{x}))$;
13: **end if**
14: $\zeta \leftarrow \lfloor \frac{73794}{27011}\zeta \rfloor$; （$\lfloor \cdot \rfloor$ は，対象の数値以下の最大の整数を表す）
15: **end while** （補注: 数回程度で成功しない場合，他のアルゴリズムに切り替える実装が多い）

2.1 計算手順

アルゴリズム 1 に基づいて，GCDHEU の計算手順を概説する。まず，計算対象である k 変数多項式の一つの変数に大きな整数 ζ を代入することで，$(k-1)$ 変数多項式の GCD 計算に問題を書き換える。この操作を再帰的に行うことにより，変数の数を一つずつ減らしていき，最後は整数 GCD の計算に帰着させる。そして，代入手続きの逆操作として，代入に用いた整数 ζ を基数とする ζ 進数として除算のみで補間して一つの変数を復元する。この操作を再帰的に行うことにより，k 変数多項式としての GCD が求められる。

例 1 (GCDHEU の計算例)
実際の計算の流れを次の多項式で示す。アルゴリズム 1 の再帰的な操作の各段階を入れ子構造で示しておくが，第 5 行目は *SymPy* で用いられている「$\zeta \leftarrow 2 \cdot \min(\|f\|_\infty, \|g\|_\infty) + 29$」を用いて計算している。

$$f(x, y) = x^2 - y^2,\ g(x, y) = x^2 - 2xy + y^2$$

1. x に $\zeta_1 = 31$ を代入し，$f(y) := f(\zeta_1, y) = -y^2 + 961$, $g(y) := g(\zeta_1, y) = y^2 - 62y + 961$ を得る。
   (a) y に $\zeta_2 = 1951$ を代入し，$f := f(\zeta_2) = -3805440$, $g := g(\zeta_2) = 3686400$ を得る。
      i. 二つの整数の *GCD* を計算し，$\gcd(-3805440, 3686400) = 3840$ を得る。
   (b) y に代入した整数 $\zeta_2 = 1951$ を用いて，$h(\zeta_2) = 3840$ となる多項式 $h(y)$ を復元する。まず，3840 から定数項 y^0 の係数は，$\mathrm{rem}(3840, 1951) = -62$ であり $h(y) \leftarrow -62$ となる。次に，$(3840 - (-62))/1951 = 2 \neq 0$ となり，y^1 の項が存在し，$\mathrm{rem}(2, 1951) = 2$ であり $h(y) \leftarrow h(y) + 2y = 2y - 62$ となる。そして，$(2-2)/1951 = 0$ となり，変数 y の復元が完了する。
2. $h(y) = 2y - 62$ の各係数において，x に代入した整数 $\zeta_1 = 31$ を用いて，$h(\zeta_1, y) = 2y - 62$ となる多項式 $h(x, y)$ を復元する。まず，定数項 x^0 の係数は，$\mathrm{rem}(2y - 62, 31) = 2y$ であり $h(x, y) \leftarrow 2y$ となる。次に，$((2y-62) - 2y)/31 = -2 \neq 0$ となり，$\mathrm{rem}(-2, 31) = -2$ であり $h(x, y) \leftarrow h(x, y) - 2x = -2x + 2y$ となる。そして，$(-2 - (-2))/31 = 0$ となり，変数 x の復元が完了する。
3. $-2x + 2y$ の原始的部分を取り出して，最終的に $\gcd(f(x, y), g(x, y)) = -x + y$ を得る。

2.2 GCD 候補の正誤判定

GCDHEU は確率的アルゴリズムであり，計算途中で求まる $\mathrm{pp}(h(\vec{x}))$ が正しい GCD ではない可能性がある（つまりは失敗する可能性がある）。それ故，試し割りにより正誤判定を行う。試し割りのみで判定可能なことは，$\zeta \geq 2 \cdot \min(\|f\|_\infty, \|g\|_\infty) + 2$ という条件下において保証されている [CGG84, Theorem 3]。計算途中の GCD 候補が正しいものではなかったとき（失敗したとき）は，代入する整数 ζ を更新（第 14 行目）し，再帰計算を継続する。なお，失敗する確率は ζ を無作為に選択する整数範囲の大きさに対して，狭義に減少することが示されている [CGG84, Theorem 5]。

3 ζ の大きさと選び方について

ζ の満たすべき条件は，前述の $\zeta \geq 2 \cdot \min(\|f\|_\infty, \|g\|_\infty) + 2$ のみである。そのため，ζ について正確にどの値を取るべきかは決まっていない。実際に論文によっても実際に使用される値が異なっており，このことが本報告における研究の発端となっている。ζ の値は各代入対象の変数においてはじめて設定するときに決定され，その変数での結果が失敗したときにより大きな値へと更新される。

具体的に ζ の大きさや選び方については，三つの観点から議論することができる。一つ目は計算量である。ζ を代入し整数演算を行う事から，より小さい値をとれば，1 回あたりの計算量は低下する。二つ目は，失敗する確率である。前述の通り，ζ を大きくしていけば，漸近的に失敗する確率が下がることがわかっている [CGG84, Theorem 5]。三つ目が，失敗したときの更新方法であり，連続して失敗しにくいような値に ζ を更新する必要がある。これに対しては，黄金比のような特別な比が提案され，使われている（第 14 行目で使われている有理数など）。

3.1 各論文における ζ の初期値の違い

前述のように，ζ の初期値は多様である。ここでは，実際に SymPy で採用されている値（SymPy と表記），Maple（Maple 2020 で確認）で採用されている値（Maple と表記），論文 [CGG89, CGG84] で採用されている値（CGG89 と CGG84 と表記）の比較を行う。なお，$\mathrm{lc}(p)$ で多項式 $p(\vec{x})$ の主係数を表す。

SymPy $\zeta = \max\left(\min\left(B, \left\lfloor 99\sqrt{B} \right\rfloor\right), 2 \cdot \min\left(\left\lfloor \frac{\|f\|_\infty}{|\mathrm{lc}(f)|} \right\rfloor, \left\lfloor \frac{\|g\|_\infty}{|\mathrm{lc}(g)|} \right\rfloor\right) + 4\right),\ B = 2 \cdot \min(\|f\|_\infty, \|g\|_\infty) + 29$

Maple $\zeta = \max\left(\min\left(B, \left\lfloor \frac{10000\sqrt{B}}{101} \right\rfloor\right), 2 \cdot \min\left(\left\lfloor \frac{\|f\|_\infty}{|\mathrm{lc}(f)|} \right\rfloor, \left\lfloor \frac{\|g\|_\infty}{|\mathrm{lc}(g)|} \right\rfloor\right) + 2\right),\ B = 2 \cdot \min(\|f\|_\infty, \|g\|_\infty) + 29$

CGG89 $\zeta = 2 \cdot \min(\|f\|_\infty, \|g\|_\infty) + 2$

CGG84 $\zeta = 7 \cdot \min(\|f\|_\infty, \|g\|_\infty) + 1$

SymPy と Maple では複雑な式が見られるが，$B = 2 \cdot \min(\|f\|_\infty, \|g\|_\infty) + 29$ の部分が基本となる。

3.2 異なる初期値に関する比較実験

簡単ではあるが，これらの ζ の初期値について，計算時間の比較実験を行った。実験に用いた多項式は Liao と Fateman[LF95] が用いたものであり，その特徴として GCDHEU が失敗しないということがあげられる。実験の結果，CGG89 の初期値を用いた場合が最も速かった。アルゴリズムが失敗することがないことから，ζ の最小値をとっている CGG89 が最も速くなることは当然の結果と言える。また，無作為に選んだ多項式を用いて追加実験した範囲においては，失敗することも少なかった。

3.3 理想的な ζ の選択

前述の実験からも，失敗しない限りにおいては，ζ の値が小さいほど計算が速いことが想像できる。そのため，小さい ζ の値で計算を進めるため，Monagan と Wittkopf[MW00] や Davenport と Padget[DP85] は，ひとまず小さい値で互いに素か判定してから大きくするなどの提案をしている。よって，ζ の理想的な値は，なるべく ζ の値を小さく抑えつつも失敗しにくい値ということになる。なお，本報告では，次章で扱う変数評価順に着目した改善を行っている関係もあり，ζ の理想的な値の具体的な議論はここまでに留める。

4 変数の評価順とその改善

前述の内容から計算過程において ζ の値を可能な限り小さく抑えることが良いと考えられる。ζ の値は変数ごとに定められるため，変数の評価順によってそれぞれの ζ の最小値が変化する可能性がある。GCDHEU の計算手順では，各変数ごとに大きな整数である ζ を代入し変数を一つずつ減らし，最後は整数 GCD に帰着させている。各変数ごとに大きな整数 ζ を代入していく性質上，整数 GCD の計算に用いられる整数の大きさは計算コストに多大な影響を与える。つまり，変数の評価順によって ζ の最小値が変化することを用いて，整数 GCD の計算コストを抑えることが可能ではないかということが本報告での問題提起である。

例 2 (変数の評価順と整数の大きさの変化)
実際の計算において必要となる整数 *GCD* を求める対象の整数の大きさの変化を次の多項式で考える。アルゴリズム *1* の第 *5* 行目は *SymPy* の初期値「$\zeta \leftarrow 2 \cdot \min(\|f\|_\infty,\ \|g\|_\infty) + 29$」を用いて計算している。

$$f(x,y,z) = x^{50} + y^{50} + z^{100},\ g(x,y,z) = x^{50} + y^{50} + z^2$$

1. (x,y,z) の評価順のとき
   (a) x に $\zeta_1 = 31$ を代入し，変数が減った次の多項式を得る。
   $$f(y,z) := f(\zeta_1,y,z) = \zeta_1^{50} + y^{50} + z^{100},\ g(y,z) := g(\zeta_1,y,z) = \zeta_1^{50} + y^{50} + z^2$$
   (b) y に $\zeta_2 = 2 \cdot \min(\zeta_1^{50},\ \zeta_1^{50}) + 29 \approx 7.4 \times 10^{74}$ を代入し，変数が減った次の多項式を得る。
   $$f(z) := f(\zeta_1,\zeta_2,z) = \zeta_1^{50} + \zeta_2^{50} + z^{100},\ g(z) := g(\zeta_1,\zeta_2,z) = \zeta_1^{50} + \zeta_2^{50} + z^2$$
   (c) z に $\zeta_3 = 2 \cdot \min(\zeta_1^{50} + \zeta_2^{50},\ \zeta_1^{50} + \zeta_2^{50}) + 29 \approx 5.7 \times 10^{3743}$ を代入し，次の整数を得る。
   $$f(\zeta_1,\zeta_2,\zeta_3) = \zeta_1^{50} + \zeta_2^{50} + \zeta_3^{100} \approx 4.7 \times 10^{374375},\ g(\zeta_1,\zeta_2,\zeta_3) = \zeta_1^{50} + \zeta_2^{50} + \zeta_3^2 \approx 3.3 \times 10^{7487}$$
   (d) 約 4.7×10^{374375} と約 3.3×10^{7487} という巨大整数の *GCD* を計算する必要がある。
2. (z,y,x) の評価順のとき
   (a) z に $\zeta_1 = 31$ を代入し，変数が減った次の多項式を得る。
   $$f(x,y) := f(x,y,\zeta_1) = x^{50} + y^{50} + \zeta_1^{100},\ g(x,y) := g(x,y,\zeta_1) = x^{50} + y^{50} + \zeta_1^2$$
   (b) y に $\zeta_2 = 2 \cdot \min(\zeta_1^{100},\ \zeta_1^2) + 29 = 1951$ を代入し，変数が減った次の多項式を得る。
   $$f(x) := f(x,\zeta_2,\zeta_1) = x^{50} + \zeta_2^{50} + \zeta_1^{100},\ g(x) := g(x,\zeta_2,\zeta_1) = x^2 + \zeta_2^{50} + \zeta_1^2$$
   (c) x に $\zeta_3 = 2 \cdot \min(\zeta_2^{50} + \zeta_1^{100},\ \zeta_2^{50} + \zeta_1^2) + 29 \approx 6.5 \times 10^{164}$ を代入し，次の整数を得る。
   $$f(\zeta_3,\zeta_2,\zeta_1) = \zeta_3^{50} + \zeta_2^{50} + \zeta_1^{100} \approx 5.0 \times 10^{8240},\ g(\zeta_3,\zeta_2,\zeta_1) = \zeta_3^{50} + \zeta_2^{50} + \zeta_1^2 \approx 5.0 \times 10^{8240}$$
   (d) 約 5.0×10^{8240} と約 5.0×10^{8240} という巨大整数の *GCD* を計算する必要がある。

これらの結果を比較すると，変数の代入順序によって整数 GCD の対象となる整数の大きさが大きく異なっていることがわかる。その原因は変数の次数差にある。次数差のある変数 z に着目すると，z が後者の評価順のように前の方で評価された場合，$\zeta_1 = 2\min(\|f\|_\infty, \|g\|_\infty) + 29$ から，小さい次数側 g のノルムが用いられ，大きい次数側 f は無視される。しかし，後者の評価順のように最後に評価された場合，大きい次数側は無視されず，そのまま整数の大きさに影響する。よって，可能な限りより小さい整数を扱うこととなる後者の評価順を選択すべきである。

4.1 変数の評価順と整数の大きさの関係に基づくアルゴリズムの改善

変数の評価順と整数の大きさの関係についてまとめる。3 変数における ζ と整数の大きさは，SymPy の値である 29 を採用した場合，次の関係によって定まる。

$$\underline{\zeta_1} = 2 \cdot \min(\|f(x,y,z)\|_\infty,\ \|g(x,y,z)\|_\infty) + 29$$

$$\underline{\zeta_2} = 2 \cdot \min(\|f(\zeta_1,y,z)\|_\infty,\ \|g(\zeta_1,y,z)\|_\infty) + 29$$

$$\underline{\zeta_3} = 2 \cdot \min(\|f(\zeta_1,\zeta_2,z)\|_\infty,\ \|g(\zeta_1,\zeta_2,z)\|_\infty) + 29$$

$$\gcd(f(\zeta_1,\zeta_2,\zeta_3),\ g(\zeta_1,\zeta_2,\zeta_3))$$

変数がそれぞれある程度の次数を持つ場合，ζ を代入した部分が大きな整数を持つ係数となる単項式となり，最大次数の項の係数の絶対値が多くの場合無限大ノルムとなる。つまり，ζ_1 を多項式 f の変数に代入後，その最大次数が n ならば，その多項式 f の無限大ノルムは約 ζ_1^n となる。また ζ を定める際，二つの多項式について無限大ノルムの大小を判定している。これらの性質を元に，例えば 3 変数 (x,y,z) の次数がそれぞれ，(a,b,c), (ℓ,m,n) である多項式 f, g を考える。このとき，$a < \ell$, $b < m$, $c < n$ と仮定し、(x,y,z) の順に評価するとする。まず，ζ_1 はそれぞれの無限大ノルムの小さい方に依存して定まる。次に，ζ_2 は無限大ノルムの小さい方に依存して定まり，$a < \ell$ より，多くの場合 ζ_2 は約 ζ_1^a となる。これより ζ_2 は，小さい次数にのみ依存していることがわかる。次に ζ_3 は，無限大ノルムの小さい方に依存して定まり，$b < m$ より，ζ_3 は約 $\zeta_2^b = \zeta_1^{ab}$ となる。最後に ζ_3 を代入した f と g はそれぞれ，$f(\zeta_1,\zeta_2,\zeta_3)$, $g(\zeta_1,\zeta_2,\zeta_3)$ となり整数だけとなっている。整数の大きさは，それぞれ f が約 $\zeta_3^c \approx \zeta_2^{bc} \approx \zeta_1^{abc}$, g が約 $\zeta_3^n \approx \zeta_2^{bn} \approx \zeta_1^{abn}$ となる。これより，変数の評価順において最後に評価された変数の次数の大きさは無限大ノルムの大小判定によって無視されず，整数の大きさに影響を与えることがわかる（ζ_1 の指数部に n が残っている）。

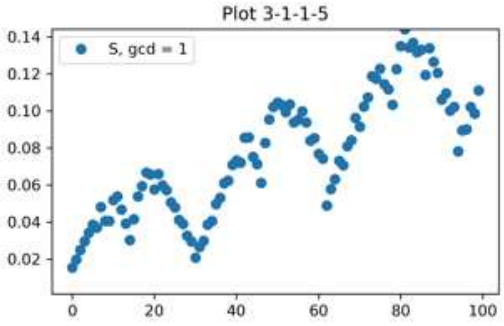
よって，変数の評価順で計算コストが変わることから，次のような手法を提案する。この方法によって，ζ がどの初期値を採用している場合でも，整数 GCD の計算コストを抑制することが期待される。

提案手法 「次数差の大きい変数を最初の方で評価し，次数差の小さい変数を最後の方で評価する」

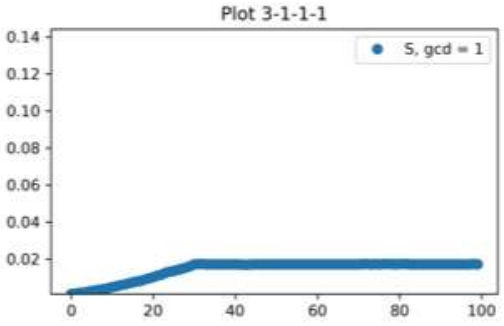
例 3 (次数差による計算時間の変化)
n を自然数とし，互いに素な疎多項式を $f(x,y,z) = x^{92} + y^{32} + z^{32}$, $g(x,y,z) = x^{92} + y^{53} + z^n$ とする[2)]。(x,y,z) の順に変数を評価した場合と，(z,y,x) の順に変数を評価した場合で次数差の影響を確認するため，n を $2 \le n \le 102$ の範囲で変化させつつ，$GCDHEU$ の計算時間を計測[3)]した結果が図 1 である。

n の変化に伴い計算時間が増大している図 1a に比べ，図 1b の計算時間が低く抑えられていることがわかる。なお，図 1b においては，途中から計算時間がおおよそ一定である（32 以上の n）。これは，$n = 102$ のように大きな n であっても次の ζ 計算時に求める $\min(\|f\|_\infty, \|g\|_\infty)$ において，より小さな値となる z^{32} の項をもつ多項式側が選ばれ，影響を及ぼさないからである。

[2)] 講演時のスライドでは，$g(x,y,z) = x^{92} + y^{32} + z^n$ と提示していたが，正しくは $g(x,y,z) = x^{92} + y^{53} + z^n$ である。
[3)] GCDHEU を 10 ループさせたものを 5 回計測し，その上位 3 つの平均時間をとったものが縦軸の値である。



(a) (x, y, z) の順に評価



(b) (z, y, x) の順に評価

図 1: 次数差による計算時間（秒）の変化

計算時間	評価順序	計算時間	評価順序	計算時間	評価順序
0.344	(w, y, x, z)	0.656	(y, z, x, w)	1.250	(x, w, z, y)
0.500	(y, x, w, z)	0.750	(z, x, y, w)	1.531	(x, y, z, w)
0.516	(x, y, w, z)	0.797	(w, x, z, y)	2.438	(y, w, z, x)
0.531	(y, w, x, z)	0.812	(z, w, x, y)	2.859	(w, y, z, x)
0.562	(w, x, y, z)	0.906	(y, x, z, w)	3.297	(w, z, y, x)
0.578	(z, x, w, y)	0.938	(w, z, x, y)	3.328	(z, w, y, x)
0.609	(x, z, w, y)	1.094	(x, w, y, z)	3.359	(y, z, w, x)
0.625	(z, y, x, w)	1.234	(x, z, y, w)	3.453	(z, y, w, x)

表 1: 変数の評価順による一回あたりの計算時間（秒）の変化

例 4 (変数の評価順による計算時間の変化)
互いに素な次の疎多項式の組について，変数の全評価順で計算時間を計測した結果が表 *1* である。

$$f(x, y, z, w) = -x^{25} - x^{11}y^2z^6w^7 - y^{33} + z^{32} - w^{30} + 1$$
$$g(x, y, z, w) = -x^7y^{10}z^8w^3 + x^6y^{10}z^7w^{11} - x^2 + y^{32} + z^{32} - w^{34}$$

表 *1* の各行では，左側の数値が計算時間を，右側が変数の評価順を表している。次数差の大きい変数 x を最後に評価している場合が，最短時間に比べて 10 倍ほど時間がかかっていることがわかる。また，次数差のない z の評価を最後に行ったときが一番速いことがわかる。

4.2 ランダム生成多項式による実験

本報告で提案している改善が計算時間の観点から効果的であるかを確認するため，ランダムに生成した多項式を用いて実験を行った。実験に用いた環境は，Python 3.12.7, SymPy1.13.2 (Ubuntu 22.04.05 LTS), Xeon E5-2687W v4 (3.0-3.5GHz), 256GB である。留意事項として，実験環境が SymPy のため ζ の初期値は SymPy の値を用いているが，他の初期値でも結果に大きな影響は与えない。また，GCDHEU で求めた結果が正しくない（失敗する）原因が整数 GCD に帰着した際にあらわれる本来の GCD の余因子部分にある [CGG89, Lemma 3] ことに加え，整数 GCD の計算コストの評価を目的にしていることから，実験では互いに素な多項式を用いる。

変数の評価順が計算速度に与える影響について，次の 4 つの方法（評価順）を調べる。まず，Plan0 では純粋に多項式環の定義に現れる変数の順に評価を行う。次に，Plan1 では次数差の大きい変数を $n-1$ 番目に評価し，他の変数は次数差の大きい変数から（同着時は小さい方の次数の小さい順で）評価を行う。Plan2 では次数差の大きい変数を $n-1$ 番目に，他の変数は次数の和の大きい順（同着時は差の小さい順）に評価を行う。Plan3 では次数差の大きい変数から評価を行う。この中で，既存の方法が Plan0 であり，今回提案した手法が Plan3 となる。

実験に用いた多項式は次の条件で生成した。共通する条件として，4 変数多項式 1000 組を 4 グループ（計 4000 組）生成した。各係数は $[-100, 100] \in \mathbb{Z}$ からランダムにとり，係数が 0 でない単項式がその全次数に関して，全次数 50 以下で一様ランダムに生成する。なるべく互いに素にするため，$f(x, y, z, w)$ は定数項を持たせ，$g(x, y, z, w)$ は定数項を持たせていない。また，疎の度合いを比べるため，多項式に含まれる項数をグループ毎に 10 個，100 個，1000 個，2000 個とした。

これらについて，同一計算 20 回分の計算時間で Plan に関係なく，全ての変数順序（24 通り）で計測を行った。表 2 と図 2 が結果となる。

	Plan0	Plan1	Plan2	Plan3
項数: 10	0.996260 (0.120116)	0.912457 (0.081040)	0.947957 (0.094860)	0.902398 (0.081842)
項数: 100	0.999309 (0.127280)	0.961142 (0.105976)	0.980153 (0.110285)	0.953730 (0.099405)
項数: 1000	0.994412 (0.152757)	0.983115 (0.138063)	1.002760 (0.143345)	0.984228 (0.129248)
項数: 2000	0.997711 (0.135142)	0.992425 (0.121321)	0.999963 (0.126710)	0.984545 (0.111340)
トータル	0.996923 (0.134336)	0.962284 (0.117651)	0.982708 (0.122102)	0.956225 (0.111958)

表 2: 全評価順の平均計算時間（秒）に対する比の平均（下段は標本標準偏差）

表 2 の値は，上段が全評価順の平均計算時間に対する比の平均，下段は標本標準偏差を表している。比の平均であり，1.0 より値が大きいほど遅く，小さいほど速いものとなる。項数 10 のときの Plan0 と Plan3 を比べると，Plan0 の値が約 1.0 なのに対し，Plan3 は約 0.9 と 10%ほど速いことがわかる。この傾向は，疎の度合いが低下するにあたり差は縮まっていくものの，トータルで見た場合，Plan3 は既存の方法に対し，4%程速いことがわかる。図 2 は同じ結果であるが，左図は比の常用対数による散布図であり，右図はそれを計測時間の小さい順にソートした散布図である。これらの結果から，提案した手法である Plan3 は既存の方法に対し改善の効果が出ていることがわかる。

5 まとめ

現在，GCDHEU の実行計算測度の改善に取り組み中であり，特にプラットフォームとして SymPy を用いている。変数の評価順による改善については次数差の小さい変数を最後に評価することにより，ζ のどの初期値でも，整数 GCD の計算コストを抑制するこに成功している。ζ の初期値や更新方法の現在の環境における視点での改善は模索中である。注意として，SymPy の多項式表現は分散型であり，変数の評価順の

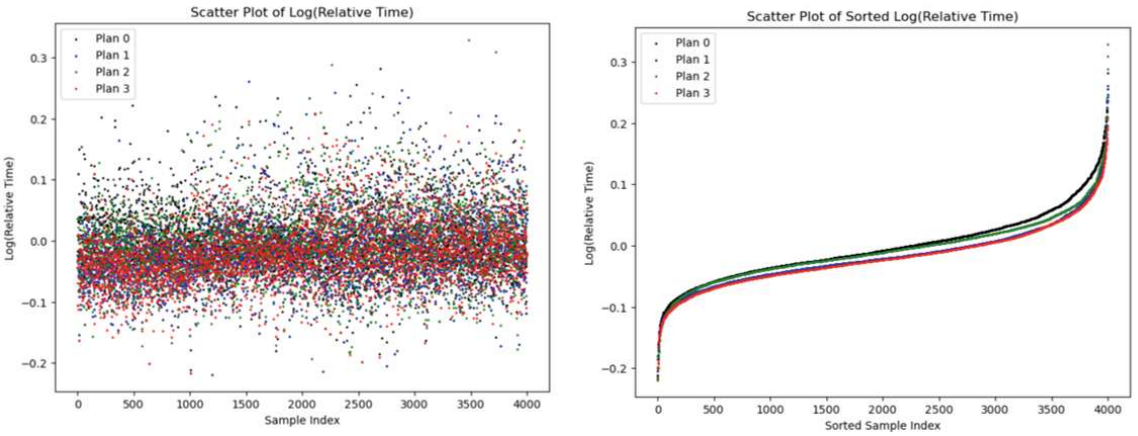


図 2: 平均計算時間（s）に対する比の常用対数散布図（左）と計算時間順でソートした散布図（右）

変更が評価コストに影響を与えない。そのため今後の展望として，SymPy 以外の GCDHEU にも効果があるか検証したいと考えている。

参　考　文　献

[Bro78] William S Brown. The subresultant PRS algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 4(3):237–249, 1978.

[CGG84] Bruce W. Char, Keith O. Geddes, and Gaston H. Gonnet. GCDHEU: heuristic polynomial GCD algorithm based on integer GCD computation. In *EUROSAM '84 (Cambridge, 1984)*, volume 174 of *Lecture Notes in Comput. Sci.*, pages 285–296. Springer, Berlin, 1984.

[CGG89] Bruce W. Char, Keith O. Geddes, and Gaston H. Gonnet. GCDHEU: heuristic polynomical GCD algorithm based on integer GCD computation. *J. Symbolic Comput.*, 7(1):31–48, 1989.

[DP85] James Davenport and Julian Padget. HEUGCD: how elementary upperbounds generate cheaper data. In *EUROCAL '85 (Linz, 1985)*, volume 204 of *Lecture Notes in Comput. Sci.*, pages 18–28. Springer, Berlin, 1985.

[LF95] Hsin-Chao Liao and Richard J. Fateman. Evaluation of the heuristic polynomial gcd. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation*, ISSAC '95, page 240–247, New York, NY, USA, 1995. Association for Computing Machinery.

[MW00] Michael B. Monagan and Allan D. Wittkopf. On the design and implementation of brown's algorithm over the integers and number fields. In *Proceedings of the 2000 International Symposium on Symbolic and Algebraic Computation*, ISSAC '00, page 225–233, New York, NY, USA, 2000. Association for Computing Machinery.

[Zip79] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM '79 (Marseille, 1979)*, volume 72 of *Lecture Notes in Comput. Sci.*, pages 216–226. Springer, Berlin, 1979.